Problem Set 2 solution

Data Structures and Algorithms 2 - Fall 2023 Due September 28, 2023 at 11:59 PM

PROBLEM 1 *Solving Recurrences*

Prove a (as tight as possible) O (big-Oh) asymptotic bound on the following recurrences. You may use any base cases you'd like.

1. For the following two recurrences, it may be helpful to draw out the tree. However, you should prove the asymptotic bound using induction.

   - $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n$

     Solution: To show: $T(n) \leq 8n$,
     base case: $T(1) = 1 \leq 8$,
     inductive hypothesis: $\forall n \leq n_0, T(n) \leq 8n$,
     inductive step: $T(n_0+1) \leq 8(n_0+1)$

$$\begin{aligned} T(n_0+1) &= T(\frac{n_0+1}{2}) + T(\frac{n_0+1}{4}) + T(\frac{n_0+1}{8}) + (n_0+1) \\ &\leq 8(\frac{n_0+1}{2}) + 8(\frac{n_0+1}{4}) + 8(\frac{n_0+1}{8}) + (n_0+1) \\ &= 4(n_0+1) + 2(n_0+1) + 1(n_0+1) + (n_0+1) \\ &= 8(n_0+1) \end{aligned}$$

     Therefore $T(n) = O(n)$. Finally, since $T(n) \geq n$, we have $T(n) = \Omega(n)$, so $T(n) = \Theta(n)$.

   - $T(n) = 2T(\frac{n}{3}) + T(\frac{n}{6}) + n$

     Solution:$T(n) \in \Theta(n)$. To show: $T(n) \leq 6n$,
     base case: $T(1) = 1 \leq 6$,
     inductive hypothesis: $\forall n \leq x_0, T(n) \leq 6n$,
     inductive step: $T(x_0+1) \leq 6(x_0+1)$

$$\begin{aligned} T(x_0+1) &= 2T(\frac{x_0+1}{3}) + T(\frac{x_0+1}{6}) + (x_0+1) \\ &\leq 12(\frac{x_0+1}{3}) + 6(\frac{x_0+1}{6}) + (x_0+1) \\ &= 4(x_0+1) + 1(x_0+1) + (x_0+1) \\ &= 6(x_0+1) \end{aligned}$$

     Therefore $T(n) \in O(n)$. Finally, since $T(n) \geq n$, we have $T(n) \in \Omega(n)$, so $T(n) \in \Theta(n)$.

2. For the following recurrence relations, indicate: (i) which case of the Master Theorem applies (if any); (ii) justification for why that case applies (if one does) i.e., what is a, $f(n)$, ε, etc; (iii) the asymptotic growth of the recurrence (if any case applies).

   - $T(n) = 7T(\frac{n}{5}) + n \log n$

     Solution: Master Theorem: Case 1, so $\Theta(n^{log_5 7})$
   - $T(n) = 3T(\frac{n}{3}) + n \log n$

     Solution: Master Theorem: Case 3 seems to apply, but not possible to find an ϵ

Solution:

PROBLEM 2 *Climate History*

Scientists call *paleoclimatologists* study the history of climate on earth before instruments were invented to measure temperatures, precipitation, etc. They try to reconstruct climate history using data found from analyzing rocks, sediments, tree rings, fossils, ice sheets, etc.

A group is trying to better understand periods of low precipitation (e.g. dryness) for a time period that spans many thousands of years. Using various data sources, they have assigned a value d_i for the relative dryness for each time-unit (let's say it's measured for every century) for this very long time period. For each time-unit i, one of five values has been assigned as d_i:

- -3 for very high precipitation
- -1 for high precipitation
- 0 for normal (or unknown)
- 1 for low precipitation
- 3 for very low precipitation

The scientists want to find the score for the driest period of consecutive years in their data. Because the effects of a drought are cumulative, the score for a period starting at time i and ending at time j will be the value when all scores from i and j are added. Consider this example with $n = 16$:

d_i	0	3	3	0	-3	-3	1	0	3	-1	-1	0	1	1	3	0
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

The period [1:2] looks high with a score of 6. But the highest score is 7 for this period [6:14]. Note there are some negative values in the middle of this period, but the high values around those make it worth having a period that includes those negative values. (The period [6:15] also has score 7. There can be more than one period with the largest score, but you just need to return the highest score, so the existence of multiple periods with the same best score doesn't matter.)

There are many ways to solve this problem, including a $\Theta(n^2)$ brute-force approach that calculates scores for all combinations of starting and ending values for a period and keeps track of the largest score. You need to do better than that. (Though if you were ever to code your solution, coding the brute-force approach is a good way to test your algorithm. Coding is **not** required for this problem.)

What you need to do: Describe an algorithm (clearly in words or in pseudocode) that uses a divide and conquer algorithm that solves this problem in $\Theta(n \log n)$ time. Your description should make it clear what the base case is, what work is done in dividing and in combining, and what recursive subproblems are solved. Along with your algorithm's description, give a brief but convincing argument that the time complexity is $\Theta(n \log n)$.

The inputs to your algorithm will be n, the number of samples, and a list d that stores n values, where each is either -3, -1, 0, 1 or 3. Your algorithm will output the largest dryness score for a period in the data (as described above).

Solution:

This is a very slight variant of the *maximum sum subarray problem*. (In that problem, the values can be any value, not just the 5 possible values for dryness.)

Here's the strategy:

1. Base Case: If the input list d has only one element, return that item as the result, which is the dryness score for that single time unit.
2. Divide: Split the input list d into two equal-sized sublists, d_{left} and d_{right}.
3. Recursive subproblems: Recursively find the solution for each of the two sublists:
   - `max_left`: The maximum score of a period that is totally contained in d_{left}.
   - `max_right`: The maximum score of a period that is totally contained in d_{right}.
4. Combine: First, we need to find the maximum dryness score that spans across the division between d_{left} and d_{right}.
   - `max_crossing`: The maximum score of a period that spans the midpoint and includes values in both d_{left} and d_{right}. This is the sum of two values calculated from the midpoint towards both ends: the maximum score from the end of d_{left} back towards the front, and then the maximum score from the beginning of d_{right} towards the end.

   Then, the maximum score for the entire list will be the maximum of these three values, i.e.: `max(max_left, max_right, max_crossing)`.

Here's Python pseudocode for the algorithm. (For efficiency reasons, we would use index values *first* and *last* and not call the Python list slice function (e.g. `d_left = d[:mid]`) since this copies list items, adding an unnecessary amount of $\Theta(n)$ work for each recursive call.)

```
def find_max_dryness_score(d):
    n = len(d)

    # Base case
    if n == 1:
        return d[0]

    # Divide
    mid = n // 2
    d_left = d[:mid]
    d_right = d[mid:]

    # Recursive calls
    max_left = find_max_dryness_score(d_left)
    max_right = find_max_dryness_score(d_right)

    # Calculate max_overlap
    max_sum_left = max_sum_right = 0
    sum_left = sum_right = 0

    # Calculate max sum starting from the end of d_left
    for i in range(mid - 1, -1, -1):
        sum_left += d[i]
        max_sum_left = max(max_sum_left, sum_left)

    # Calculate max sum starting from the beginning of d_right
    for i in range(mid, n):
        sum_right += d[i]
        max_sum_right = max(max_sum_right, sum_right)

    # Combine and return the maximum dryness score
    max_crossing = max_sum_left + max_sum_right
    return max(max_left, max_right, max_crossing)
```

PROBLEM 3 *Fast Transformations*

Computer graphics software typically represents points in n dimensions as $(n+1)$-dimensional vectors. To make transformations on the points (e.g. rotating a modelled figure, zooming in, or making the figure appear as though seen through a fish-eye lens) we use a $(n+1) \times (n+1)$ matrix which defines the transformation, and then we multiply each vector by this matrix to transform that point. Let's say we are developing software for very high dimension graphics (n dimensions), and we have a transformation that we would like to apply to a particular point n times. Develop an algorithm which can multiply this $(n+1)$-dimensional point by this $(n+1) \times (n+1)$ transformation matrix n times in $o(n^3)$ (little-oh of n^3) time. Prove this run time.

Solution:

The n^{th} transform of the vector v using transformation matrix T is given by $T(T(T...T(v))))$, where T is multiplied by v n times, i.e. it is $T^n \cdot v$. This means we can use the algorithm from part 1 to first calculate T^n with Strassen's algorithm being used for each matrix multiplication, which gives a run time of $O(n^{\log_2 7} \log n)$.

Note that $\forall a > b > 0, \log(n) = o(n^a)$ and $n^b = o(n^a)$,

therefore $n^{\log_2 7} \log n = o(n^{2.9} \cdot n^{0.1}) = o(n^3)$ as $n^{log_2 7} = o(n^{2.9})$ and $\log n = o(n^{0.1})$.

PROBLEM 4 *Receding Airlines*

You have been hired to plan the flights for Professor Floryan's brand new passenger air company, "Receding Airlines." Your objective is to provide service to n major cities within North America. The catch is that this airline will only fly you East.

You recognize that in order to enable all your passengers to travel from any city to any other city (to the East) with a single flight requires $\Omega(n^2)$ different routes. Prof. Floryan says that the airline cannot be profitable when supporting so many routes. Another option would be to order the cities in a list (from West to East), and have flights that go from the city at index i, to the city at index $i+1$. This requires $\Theta(n)$ routes, but would mean that some passengers would require $\Omega(n)$ connections to get to their destination.

1. Devise a compromise set of routes which requires no passenger have more than a single connection (i.e. must take at most two flights), and requires no more than $O(n \log n)$ routes. Prove that your set of routes satisfies these requirements.

2. After a few years, passengers start demanding routes from East to West, and you decide to support new routes from East to West (in addition to supporting routes from West to East). Show that with routes in *both* directions, it is possible to connect all n cities with just $O(n)$ routes such that no passenger needs more than a single connection to get to their destination.

Note: In class so far, we've used recurrence relations to count an algorithm's time complexity, i.e. how many basic operations are executed. Here we're using one to count the number of flights. While this is a bit different than measuring an algorithm's time-complexity, we can still use a divide and conquer approach and a recurrence relation to answer these questions.

Solution:

Base case: $n = 1$, in this case just return the city.

Otherwise, split the cities into two halves, a West half and a East half, using the median West-East city. If there were an odd number of cities then exclude the median city from both halves. Since each half was solved recursively, we know that it is possible to go from any city in the East sublist of cities to any other city within that same sublist to its West (the same applies within the West sublist). Therefore, we now only need to add routes that allow travel from any city in the East sublist to any city in the West sublist using at most two hops. To do this, we will add a route from every city in the East sublist to either the center city (in the case that n was odd) or the West-most city in the East sublist (in the case that n was even), then add a connection from that city to every city in the West sublist.

The number of routes drawn using this algorithm on n cities ($T(n)$) can be derived by counting the number of routes drawn in the two recursive subcalls ($T(\frac{n}{2})$ each), plus the number needed to combine the sub-solutions $n - 1$. This gives the recurrence $T(n) = 2T(\frac{n}{2}) + n - 1$. Case 2 of the Master Theorem applies here, giving a total of $\Theta(n \log n)$ routes.

problem 5 *Bazinga!*

Theoretical Physicist Sheldon Cooper has decided to give up on String Theory in favor of researching Dark Matter. Unfortunately, his grant-funded position at Caltech is dependent on his continued work in String Theory, so he must search elsewhere. He applies and receives offers from MIT and Harvard. While money is no object to Sheldon, he wants to ensure he's paid fairly and that his offers are at least the median salary among the two schools' Physics departments. Therefore, he hires you to find the median salary across the two departments. Each school mantains a database of all of the salaries for that particular school, but there is no central database.

Each school has given you the ability to access their particular data by executing *queries*. For each query, you provide a particular database with a value k such that $1 \leq k \leq n$, and the database returns to you the k^{th} smallest salary in that school's Physics department.

You may assume that: each school has exactly n physicists (i.e. $2n$ total physicists across both schools), every salary is unique (i.e. no two physicists, regardless of school, have the same salary), and we define the *median* as the n^{th} highest salary across both schools.

1. Design an algorithm that finds the median salary across both schools in $\Theta(\log(n))$ total queries.

2. State the complete recurrence for your algorithm. You may put your $f(n)$ in big-theta notation. Show that the solution for your recurrence is $\Theta(\log(n))$.

3. Prove that your algorithm above finds the correct answer. *Hint: Do induction on the size of the input.*

Solution:

Solution: Algorithm description:

1. Let s be the start index (initially 1) and e be the end index (initially n) of interest. These are given as parameters to our function (note that there are actually two of each (one for MIT and one for Harvard).

2. choose the $k = \frac{(s+e)}{2}$ highest salary from each respective database (technically you want the $k+1$ salary if $e-s$ is even and the $\lceil k \rceil$ if $e-s$ is odd). Note that these are the median salaries for each respective university. Let's call these values A and B.
3. Compare A and B (and suppose that $A < B$. We are going to ignore the half of the first list with values below A and the values in the second list with values greater than B. Notice that if $e-s$ is even, we need to remove B as well (see proof below for reasoning).
4. recursively call find median with left indices set to $s = 1$ and $e = k$ and the right indices set to $s = k$ and $e = n$. Note that if $e-s$ was even, then the right call has $s = k+1$.
5. the base case is when there are $n = 1$, and we query the database twice and simply select the larger of the two items as our median.

Solution: Recurrence: $T(n) = T(\frac{n}{2}) + \Theta(1)$. This solves to $\Theta(log(n))$ via the master theorem case 2. **Solution:** Proof of correctness: We do this by induction on the size of the lists.

Base case is $n = 1$, and we know the median is the larger of the two items, and that is what the base case of the algorithm returns! Base case proven!
I.H. We then assume the algorithm works (finds the correct median) for all $i \leq k-1$.
I.S. We must then show that it works for a list of size k. To do this we show two things:

1. The list does not remove the median in the divide step
2. the median of the new half-sized list is the same median as the original list (thus our recursive call, by inductive hypothesis, will find it correctly).

For the first one, we make sure that what we remove is not the median. We do this by using the fact that if at least half the total list size (k in this case) is greater then (or less than) the items you remove, then they do not contain the median (notice that for the upper items you remove, at least $k+1$ items must be less than those items). You define A and B as the medians of the respective lists. Assume that $A < B$, and we count the items greater than or equal to A. We know that the $\frac{k}{2}$ items above and including A in the first list are in this category, and because $A < B$, the $\frac{k}{2}$ items above and including B, are also in this category. Thus, the items below A do not contain the median (and by similar argument, the items above B as well).

For the second part (from the list above), we must have removed the same number from each end of the list (this ensures that the median of the new list is the same as the median of the whole list). If k is odd, this is not an issue (we remove $\lfloor \frac{k}{2} \rfloor$ from each side). If k is even, then we throw out the $\frac{k}{2}$ items exactly below A, and the $\frac{k}{2} - 1$ items above B along with B. Thus, the median must be in the new list and by the inductive hypothesis, the algorithm works for size k.

PROBLEM 6 *Mission Impossible*

As the newly-appointed Secretary of the Impossible Missions Force (IMF), you have been tasked with identifying the double agents that have infiltrated your ranks. There are currently n agents in your organization, and luckily, you know that the majority of them (i.e., strictly more than $n/2$ agents) are loyal to the IMF. All of your agents know who is loyal and who is a double agent. Your plan for identifying the double agents is to pair them up and ask each agent to identify whether the other is a double agent or not. Agents loyal to your organization will always answer honestly while double agents can answer arbitrarily. The list of potential responses are listed below:

Agent 001	Agent 002	Implication
"002 is a double agent"	"001 is a double agent"	At least one is a double agent
"002 is a double agent"	"001 is loyal"	At least one is a double agent
"002 is loyal"	"001 is a double agent"	At least one is a double agent
"002 is loyal"	"001 is loyal"	Both are loyal or both are double agents

1. A group of n agents is "acceptable" for a mission if a majority of them ($> n/2$) are loyal. Suppose we have an "acceptable" group of n agents. Describe an algorithm that has the following properties:
   - Uses at most $\lfloor n/2 \rfloor$ pairwise tests between agents.
   - Outputs a smaller "acceptable" group of agents of size at most $\lceil n/2 \rceil$.
2. Using your approach from Part 1, devise an algorithm that identifies which agents are loyal and which are double agents using $\Theta(n)$ pairwise tests.

Solution:

Part 1: To do this, we first pair up all agents, excepting for a possible odd-one-out (call this agent A_{ooo}). For a pair of agents (A_i, A_j) we will do the following:

Case 1: if either responds with "double agent", then eliminate both agents

Case 2: if the both respond with "loyal", then eliminate agent A_j.

Note that these rules will always eliminate at least $\lfloor \frac{n}{2} \rfloor$ agents, and clearly we will perform exactly $\lfloor \frac{n}{2} \rfloor$ pairwise queries. This is because Case will eliminate two of two agents, and Case will eliminate one of two. Since those cover all cases, at least half the agents will always be eliminated.

In the case that the number of loyal agents is at least the number of double agents, these rules will maintain that at least half of the agents remaining are loyal. In Case , we are guaranteed that at least one agent in the pair is double, thus we cannot eliminate more double agents than loyal agents when implementing that rule.

Case can occur only when both agents are equivalent (both loyal or both double). The situation when both agents are loyal must occur at least as often as when both agents are double. Suppose we have already eliminated all of the agents in pairs satisfying Case . Since we observed that in each pair the number of double is at least the number of loyal, that means that after all eliminations we have that the number of double eliminating is at least the number of loyal. In order to guarantee that the number of loyal we began with was at least the number of double, it must be the case that of the agents in loyal-loyal pairs, there must have been at least as many loyal as double. Since both agents in the pairs are equivalent, we must have eliminated at least as many double agents as loyal agents, and therefore will not have an double majority.

Observe that if we had a even number of agents, the above algorithm will guarantee a loyal majority of size $\frac{n}{2}$. We must now consider whether to eliminate A_{ooo}. We obey the following rules:

Case 1: If there is an even number of agents remaining apart from A_{ooo}, then keep A_{ooo}

Case 2: Otherwise eliminate A_{ooo}

Now we will show that Case guarantees a majority of loyal agents. Note that if A_{ooo} is loyal, including A_{ooo} certainly will not jeopardize a loyal majority. Since there is an even number of agents remaining, we know that the parity of remaining loyal agents is the same as the parity of the remaining double agents. In the case that A_{ooo} is double, in order to have started with a loyal majority it must be that the number of remaining loyal agents exceeds the number of remaining double agents. Because the parity must match, this implies that the number of loyal agents must be at least two more than the number of double. Thus including the double A_{ooo} does not jeopardize our loyal majority.

Now we will show that Case guarantees a majority of loyal agents. Since we assumed we started with a majority of loyal agents, it must be the number of remaining loyal agents exceeds the number of remaining double agents even without A_{ooo}, therefore excluding A_{ooo} never jeopardizes the loyal majority.

Part 2: We can use the above algorithm as the "divide" step in a divide and conquer algorithm. By doing the above, we take a set of n agents having a loyal majority, and reduce this to a subproblem in which we have a set of $\lceil \frac{n}{2} \rceil$ agents having a loyal majority. The the loyal majority invariant will hold through subsequent re-applications of the procedure, we can continuously reapply until only one agent remains. Since this set is guaranteed to have a loyal majority, that one agent must be loyal. The number of queries required by this is given by the recurrence $T(n) = T(\frac{n}{2}) + \frac{n}{2}$, which is $\Theta(n)$

proof: via induction $T(n) = T(\frac{n}{2}) + \frac{n}{2}$

to show $T(n) = O(n)$, show $T(n) \leq n$

inductive hypothesis: $\forall n \leq n_0, T(n) \leq n$

inductive step:

$$\begin{aligned} T(n_0+1) &= T(\frac{n_0+1}{2}) + \frac{n_0+1}{2} \\ &\leq \frac{n_0+1}{2} + \frac{n_0+1}{2} \\ &= n_0+1 \end{aligned}$$

to show $T(n) = \Omega(n)$, show $T(n) \geq n$

inductive hypothesis: $\forall n \leq n_0, T(n) \geq n$

inductive step:

$$\begin{aligned} T(n_0+1) &= T(\frac{n_0+1}{2}) + \frac{n_0+1}{2} \\ &\geq \frac{n_0+1}{2} + \frac{n_0+1}{2} \\ &= n_0+1 \end{aligned}$$

Now that we have identified one loyal agent, we may simply use that known loyal agent to identify the remaining agents, requiring an additional $n-1$ pairwise tests. Giving an overall $\Theta(n)$ pairwise tests required.